arXiv:1209.0488v1 [cs.RO] 3 Sep 2012

Learning Prioritized Control of Motor Primitives*

Jens Kober and Jan Peters

***Abstract*— Many tasks in robotics can be decomposed into sub-tasks that are performed simultaneously. In many cases, these sub-tasks cannot all be achieved jointly and a prioritization of such sub-tasks is required to resolve this issue. In this paper, we discuss a novel learning approach that allows to learn a prioritized control law built on a set of sub-tasks represented by motor primitives. The primitives are executed simultaneously but have different priorities. Primitives of higher priority can override the commands of the conflicting lower priority ones. The dominance structure of these primitives has a significant impact on the performance of the prioritized control law. We evaluate the proposed approach with a ball bouncing task on a Barrett WAM.**

I. INTRODUCTION

When learning a new skill, it is often easier to practice the required sub-tasks separately and later on combine them to perform the task – instead of attempting to learn the complete skill as a whole. For example, in sports sub-tasks can often be trained separately. Individual skills required in the sport are trained in isolation to improve the overall performance, e.g., in volleyball a serve can be trained without playing the whole game.

Sub-tasks often have to be performed simultaneously and it is not always possible to completely fulfill all at once. Hence, the sub-tasks need to be prioritized. An intuitive example for this kind of prioritizing sub-tasks happens during a volleyball game: a player considers hitting the ball (and hence avoiding it touching the ground and his team loosing a point) more important than locating a team mate and playing the ball precisely to him. The player will attempt to fulfill both sub-tasks. If this is not possible it is often better to "save" the ball with a high hit and hope that another player recovers it rather than immediately loosing a point.

In this paper, we learn different sub-tasks that are represented by motor primitives that combined can perform a more complicated task. For doing so, we will stack controls corresponding to different primitives that represent movements in task space. These primitives are assigned different priorities and the motor commands corresponding to primitives with higher priorities can override the motor commands of lower priority ones. The proposed approach is outlined in Sect. I-A and further developed in Sect. III. We evaluate our approach with a ball-bouncing task (see Fig. 1 and Sect. IV).

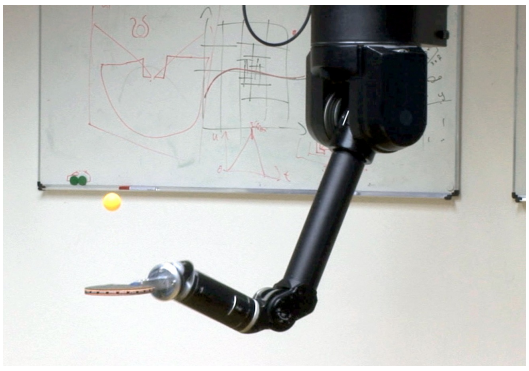


Fig. 1: This figure illustrates the ball-bouncing task on a Barrett WAM. The goal is to keep the ball bouncing on the racket.

As the sub-tasks describe the movements in task space, we have to learn a control that is mapping to the robot joint space. Unfortunately, this mapping is not a well-defined function for many robots. For example, if the considered task space has fewer degrees of freedom than the robot, multiple solutions are possible. This redundancy can be resolved by introducing a null-space control, i.e., a behavior that operates on the redundant degrees of freedom. Such a null-space control can for example pull the robot towards a rest posture [1], prevent getting close to joint limits [2], avoid obstacles [3] or singularities [4]. Computing the task space control often corresponds to an optimization problem, that can for example be solved by a gradient based approach. A well known approach is the pseudo-inverse solution [1, 3]. An alternative is to learn an operational space control law that implicitly includes the null-space behavior [5]. Once learned, it corresponds to a unique mapping from desired actions in operational space to required actions in joint space.

The problem studied in this paper is related to hierarchical control problems as discussed in [6]. Using prioritized primitives in classical control has been explored in [7] by using analytical projections into the null-space. In this paper, we propose a learning approach that does not require complete knowledge of the system, the constraints, and the task. In the reinforcement learning community, the compositions of options (i.e., concurrent options), which is related to the concurrent execution of primitives, has been studied [8]. Learning null-space control has been explored in [9]. In contrast, we do not attempt to recover the implicit null-space policy but build a hierarchical operational space control law

*The project receives funding from the European Community's Seventh Framework Programme under grant agreement no. ICT-248273 GeRT. The project receives funding from the European Community's Seventh Framework Programme under grant agreement no. ICT-270327 CompLACS.

Both authors are with the Max Planck Institute for Intelligent Systems, Department of Empirical Inference, Spemannstr. 38, 72076 Tübingen, Germany and the Technische Universität Darmstadt, Intelligent Autonomous Systems Group, Hochschulstr. 10, 64289 Darmstadt, Germany {firstname.lastname}@tuebingen.mpg.de

from user demonstrated primitives.

A. Proposed Approach

Based on the observation that many tasks can be described as a superposition of sub-tasks, we want to have a set of controls that can be executed simultaneously. As a representation for the sub-tasks, we chose the dynamical systems motor primitives, which are discussed in more detail in Section II. Such primitives are well suited as representation for the sub-tasks as they ensure the stability of the movement generation. They are invariant under transformations of the initial position and velocity, the final position and velocity, the duration as well as the movement amplitude.

In this paper, these primitives are described in different task spaces, e.g., in the form

$$\ddot{\mathbf{x}}_i = \pi_i(\mathbf{x}_i, \dot{\mathbf{x}}_i, z)$$

where z denotes a shared canonical system while $\mathbf{x}_i$ are positions in task-space i. For example, if we have a primitive "move end-effector up and down" its task space would correspond to the Cartesian position indicating the height (as well as the corresponding velocities and accelerations) but not include the sideways movement or the orientation of the end-effector. The dynamical systems motor primitives are well suited to represent different kinds of vertical movements starting and ending at various states and of different duration.

These primitives are prioritized such that

$$i \succeq i-1,$$

which reads a "task i dominates task $i-1$". If both sub-tasks can be fulfilled at the same time, our system will do so – but if this should not be possible, sub-task i will be fulfilled at the expense of sub-task $i-1$. We attempt to reproduce a complex task that consists of several sub-tasks, represented by motor primitives,

$$\{\pi_1, \pi_{2,}, \ldots, \pi_N\}$$

that are concurrently executed at the same time following the prioritization scheme

$$N \succeq N-1 \succeq \cdots \succeq 2 \succeq 1.$$

This approach requires a prioritized control law that composes the motor command out of the primitives π_i, i.e.,

$$\mathbf{u} = \mathbf{f}(\pi_1, \pi_{2,}, \ldots, \pi_N, \mathbf{q}, \dot{\mathbf{q}})$$

where $\mathbf{q}, \dot{\mathbf{q}}$ are the joint position and joint velocity, $\mathbf{u}$ are the generated motor commands (torques or accelerations).

We try to acquire the prioritized control law in three steps, which we will illustrate with the ball-bouncing task:

1) We observe $\ddot{\mathbf{x}}_i(t), \dot{\mathbf{x}}_i(t), \mathbf{x}_i(t)$ individually for each of the primitives that will be used for the task. For the ball-bouncing example, we may have the following sub-tasks: "move under the ball", "hit the ball", and "change racket orientation". The training data is collected by executing only one primitive at a time without considering the global strategy, e.g., for the "change racket orientation" primitive by keeping the position of the racket fixed and only changing its orientation without a ball being present. This training data is used to acquire the task by imitation learning under the assumption that these tasks did not need to overrule each other in the demonstration (Sect. III).
2) We enumerate all possible dominance structures and learn a prioritized control law for each dominance list that fuses the motor primitives. For the three ball-bouncing primitives there are six possible orders, as listed in Table I.
3) We choose the most successful of these approaches. The activation and adaptation of the different primitives is handled by a strategy layer (Sect. IV-B). In the ball-bouncing task, we evaluate how long each of the prioritized control laws keeps the ball in the air and pick the best performing one (Sect. IV-C).

Clearly, enumerating all possible dominance structures only works for small systems (as the number of possibilities grows with $n!$, i.e., exponentially fast).

II. BACKGROUND: MOTOR PRIMITIVES

While the original formulation in [10] for discrete dynamical systems motor primitives used a second-order system to represent the phase z of the movement, this formulation has proven to be unnecessarily complicated in practice. Since then, it has been simplified and, in [11], it was shown that a single first order system suffices

$$\dot{z} = -\tau \alpha_z z. \tag{1}$$

This canonical system has the time constant $\tau = 1/T$ where T is the duration of the motor primitive, a parameter α_z which is chosen such that $z \approx 0$ at T to ensure that the influence of the transformation function, shown in Eq. (3), vanishes. Subsequently, the internal state $\mathbf{y}$ of a second system is chosen such that positions $\mathbf{x}$ of all degrees of freedom are given by $\mathbf{x} = \mathbf{y}_1$, the velocities $\dot{\mathbf{x}}$ by $\dot{\mathbf{x}} = \tau \mathbf{y}_2 = \dot{\mathbf{y}}_1$ and the accelerations $\ddot{\mathbf{x}}$ by $\ddot{\mathbf{x}} = \tau \dot{\mathbf{y}}_2$. Under these assumptions, the learned dynamics of Ijspeert motor primitives can be expressed in the following form

$$\begin{aligned} \dot{\mathbf{y}}_2 &= \tau \alpha_y \left(\beta_y \left(\mathbf{g} - \mathbf{y}_1 \right) - \mathbf{y}_2 \right) + \tau \mathbf{A} \mathbf{f}\left(z \right), \\ \dot{\mathbf{y}}_1 &= \tau \mathbf{y}_2. \end{aligned} \tag{2}$$

This set of differential equations has the same time constant τ as the canonical system, parameters α_y, β_y set such that the system is critically damped, a goal parameter $\mathbf{g}$, a transformation function $\mathbf{f}$ and an amplitude matrix $\mathbf{A} = \mathrm{diag}(a_1, a_2, \ldots, a_n)$, with the amplitude modifier $\mathbf{a} = [a_1, a_2, \ldots, a_n]$. In [11], they use $\mathbf{a} = \mathbf{g} - \mathbf{y}_1^0$ with the initial position $\mathbf{y}_1^0$, which ensures linear scaling. Alternative choices are possibly better suited for specific tasks, see e.g., [12]. The transformation function $\mathbf{f}\left(z \right)$ alters the output of the first system, in Eq. (1), so that the second system, in Eq. (2), can represent complex nonlinear patterns and it is given by

$$\mathbf{f}\left(z \right) = \textstyle\sum_{i=1}^{N} \psi_i\left(z \right) \mathbf{w}_i z. \tag{3}$$

Here, $\mathbf{w}_i$ contains the i^{th} adjustable parameter of all degrees of freedom, N is the number of parameters per degree of freedom, and $\psi_i(z)$ are the corresponding weighting functions [11]. Normalized Gaussian kernels are used as weighting functions given by

$$\psi_i(z) = \frac{\exp\left(-h_i\left(z-c_i\right)^2\right)}{\sum_{j=1}^{N}\exp\left(-h_j\left(z-c_j\right)^2\right)}.$$

These weighting functions localize the interaction in phase space using the centers c_i and widths h_i. Note that the degrees of freedom (DoF) are usually all modeled as independent in Eq. (2). All DoFs are synchronous as the dynamical systems for all DoFs start at the same time, have the same duration, and the shape of the movement is generated using the transformation $\mathbf{f}(z)$ in Eq. (3). This transformation function is learned as a function of the shared canonical system in Eq. (1).

The original formulation assumes that the goal velocity is zero. Clearly this behavior is undesirable for hitting the balls in the ball-bouncing task. In [13], we proposed a modification that allows to specify arbitrary goal velocities:

$$\begin{aligned}
\dot{\mathbf{y}}_2 &= (1-z)\,\tau\alpha_g\left(\beta_g\left(\mathbf{g}_m-\mathbf{y}_1\right)+\frac{\left(\dot{\mathbf{g}}-\dot{\mathbf{y}}_1\right)}{\tau}\right)+\tau\mathbf{A}\mathbf{f}\\
\dot{\mathbf{y}}_1 &= \tau\mathbf{y}_2,\\
\mathbf{g}_m &= \mathbf{g}_m^0-\dot{\mathbf{g}}\frac{\ln(z)}{\tau\alpha_h},
\end{aligned}$$

where $\dot{\mathbf{g}}$ is the desired final velocity, $\mathbf{g}_m$ is the moving goal and the initial position of the moving goal $\mathbf{g}_m^0 = \mathbf{g}-\tau\dot{\mathbf{g}}$ ensures that $\mathbf{g}_m(T) = \mathbf{g}$. The term $-\ln(z)/(\tau\alpha_h)$ is proportional to the time if the canonical system in Eq. (1) runs unaltered; however, adaptation of z allows the straightforward adaptation of the hitting time.

As suggested in [10], locally-weighted linear regression can be used for imitation learning. The duration of discrete movements is extracted using motion detection and the time-constants are set accordingly. Additional feedback terms can be added as shown in [10–12].

III. LEARNING THE PRIORITIZED CONTROL LAW

By learning the prioritized control, we want to obtain a control law

$$\mathbf{u} = \ddot{\mathbf{q}} = \mathbf{f}(\pi_1, \pi_{2,}, \ldots, \pi_N, \mathbf{q}, \dot{\mathbf{q}}),$$

i.e., we want to obtain the required control $\mathbf{u}$ that executes the primitives $\pi_1, \pi_{2,}, \ldots, \pi_N$. Here, the controls correspond to the joint accelerations $\ddot{\mathbf{q}}$. The required joint accelerations not only depend on the primitives but also on the current state of the robot, i.e., the joint positions $\mathbf{q}$ and joint velocities $\dot{\mathbf{q}}$. Any control law can be represented locally as a linear control law. In our setting, these linear control laws can be represented as

$$\mathbf{u} = \begin{bmatrix}\ddot{\mathbf{x}}_i\\ \dot{\mathbf{q}}\\ \mathbf{q}\end{bmatrix}^{\mathrm{T}}\boldsymbol{\theta} = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\theta},$$

where $\boldsymbol{\theta}$ are the parameters we want to learn and $\boldsymbol{\phi} = \begin{bmatrix}\ddot{\mathbf{x}}_i & \dot{\mathbf{q}} & \mathbf{q}\end{bmatrix}$ acts as features. Often the actions of the primitive $\ddot{\mathbf{x}}_i$ can be achieved in multiple different ways due to the redundancies in the robot degrees of freedom. To ensure consistency, a null-space control is introduced. The null-space control can, for example, be defined to pull the robot towards a rest posture $\mathbf{q}_0$, resulting in the null-space control

$$\mathbf{u}_0 = -\mathbf{K}_D\dot{\mathbf{q}} - \mathbf{K}_P\left(\mathbf{q}-\mathbf{q}_0\right),$$

where $\mathbf{K}_D$ and $\mathbf{K}_P$ are gains for the velocities and positions respectively.

To learn the prioritized control law, we try to generalize the learning of the operational space control approach from [5] to a hierarchical control approach [1, 7].

A. Single Primitive Control Law

A straightforward approach to learn the motor commands $\mathbf{u}$, represented by the linear model $\mathbf{u} = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\theta}$, is using linear regression. This approach minimizes the squared error

$$\mathrm{E}^2 = \sum_{t=1}^{T}\left(\mathbf{u}_t^{\text{ref}} - \boldsymbol{\phi}_t^{\mathrm{T}}\boldsymbol{\theta}\right)^2$$

between the demonstrated control of the primitive u_t^{ref} and the recovered linear policy $\mathbf{u}_t = \boldsymbol{\phi}_t^{\mathrm{T}}\boldsymbol{\theta}$, where T is the number of samples. The parameters minimizing this error are

$$\boldsymbol{\theta} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{U}, \tag{4}$$

with $\boldsymbol{\Phi}$ and $\mathbf{U}$ containing the values of the demonstrated $\boldsymbol{\phi}$ and $\mathbf{u}$ for all time-steps t respectively, and a ridge factor λ. If the task space and the joint-space coincide, the controls $\mathbf{u} = \ddot{\mathbf{q}}$ are identical to the action of the primitive $\ddot{\mathbf{x}}_i$. We also know that *locally* any control law that can be learned from data is a viable control law [5]. The error with respect to the training data is minimized, *however*, if the training data is not consistent, the plain linear regression will average the motor commands, which is unlikely to fulfill the actions of the primitive.

In order to enforce consistency, the learning approach has to resolve the redundancy and incorporate the null-space control. We can achieve this by using the program

$$\begin{aligned}
\min_{\mathbf{u}} J &= \left(\mathbf{u}-\mathbf{u}_0\right)^{\mathrm{T}}\mathbf{N}\left(\mathbf{u}-\mathbf{u}_0\right)\\
s.t.\ \ddot{\mathbf{x}} &= \pi\left(\mathbf{x}, \dot{\mathbf{x}}, z\right)
\end{aligned} \tag{5}$$

as discussed in [1]. Here the cost J is defined as the weighted squared difference of the control $\mathbf{u}$ and the null-space control $\mathbf{u}_0$, where the metric $\mathbf{N}$ is a positive semi-definite matrix. The idea is to find controls $\mathbf{u}$ that are as close as possible to the null-space control $\mathbf{u}_0$ while still fulfilling the constraints of the primitive π. This program can also be solved as discussed in [5]. Briefly speaking, the regression in Eq. (4) can be made consistent by weighting down the error by weights w_t and hence obtaining

$$\boldsymbol{\theta} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{W}\boldsymbol{\Phi} + \lambda\mathbf{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{W}\mathbf{U} \tag{6}$$

Algorithm 1 Learning the Prioritized Control Law

define null-space control $\mathbf{u}_0$, metric $\mathbf{N}$, scaling factor α

collect controls $\mathbf{u}_{i,t}$ and features $\boldsymbol{\phi}_{i,t}$ for all primitives $i \in \{1,\dots,N\}$ and all time-steps $t \in \{1,\dots,T\}$ separately

for primitives $i = 1 \dots N$ (N: highest priority) **do**

for time-steps $t = 1 \dots T$ **do**

calculate offset controls
$\hat{\mathbf{u}}_{i,t} = \mathbf{u}_{i,t} - \sum_{j=1}^{i-1} \boldsymbol{\phi}_{i,t}^{\mathrm{T}} \boldsymbol{\theta}_j - \mathbf{u}_{0,t}$

calculate weights $\hat{w}_{i,t} = \exp\left(-\alpha \hat{\mathbf{u}}_{i,t}^{\mathrm{T}} \mathbf{N} \hat{\mathbf{u}}_{i,t}\right)$

end for

build control matrix $\hat{\mathbf{U}}_i$ containing $\hat{\mathbf{u}}_{i,1} \dots \hat{\mathbf{u}}_{i,T}$

build feature matrix $\boldsymbol{\Phi}_i$ containing $\boldsymbol{\phi}_{i,1} \dots \boldsymbol{\phi}_{i,T}$

build weight matrix $\hat{\mathbf{W}}_i = \mathrm{diag}(\hat{w}_{i,1}, \dots, \hat{w}_{i,T})$

calculate parameters
$\boldsymbol{\theta}_i = \left(\boldsymbol{\Phi}_i^{\mathrm{T}} \hat{\mathbf{W}}_i \boldsymbol{\Phi}_i + \lambda \mathbf{I}\right)^{-1} \boldsymbol{\Phi}_i^{\mathrm{T}} \hat{\mathbf{W}}_i \hat{\mathbf{U}}_i$

end for

end for

with $\mathbf{W} = \mathrm{diag}(w_1, \dots, w_{Tn})$ for T samples. This approach works well for linear models and can be gotten to work with multiple locally linear control laws. Nevertheless, it maximizes a reward instead of minimizing a cost. The cost J can be transformed into weights w_t by passing it through an exponential function

$$w_t = \exp\left(-\alpha \tilde{\mathbf{u}}_t^{\mathrm{T}} \mathbf{N} \tilde{\mathbf{u}}_t\right),$$

where $\tilde{\mathbf{u}}_t = (\mathbf{u}_t - \mathbf{u}_0)$. The scaling factor α acts as a monotonic transformation that does not affect the optimal solution but can increase the efficiency of the learning algorithm.

Using the Woodbury formula [14] Eq. (6) can be transformed into

$$\mathbf{u} = \boldsymbol{\phi}(x)^{\mathrm{T}} \boldsymbol{\Phi}^{\mathrm{T}} \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \mathbf{W}_U\right)^{-1} \mathbf{U} \tag{7}$$

with $\mathbf{W}_U = \mathrm{diag}\left(\tilde{\mathbf{u}}_1^T \mathbf{N} \tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_n^T \mathbf{N} \tilde{\mathbf{u}}_n\right)$. By introducing the kernels $\mathbf{k}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})^{\mathrm{T}} \boldsymbol{\Phi}^{\mathrm{T}}$ and $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$ we obtain

$$\mathbf{u} = \mathbf{k}(\mathbf{s})^{\mathrm{T}} \left(\mathbf{K} + \mathbf{W}_U\right)^{-1} \mathbf{U},$$

which is related to the kernel regression [15]. This kernelized form of Eq. (7) overcomes the limitations of the linear model at a cost of higher computational complexity.

B. Prioritized Primitives Control Law

In the previous section, we have described how the control law for a single primitive can be learned. To generalize this approach to multiple primitives with different priorities, we want a control law that always fulfills the primitive with the highest priority and follows the remaining primitives as much as possible according to their place in the hierarchy. Our idea is to represent the higher priority control laws as correction term with respect to the lower priority primitives. The control of the primitive with the lowest priority is learned first. This control is subsequently considered to be a baseline and the primitives of higher priority only learn the difference to this baseline control. The change between the motor commands resulting from primitives of lower priority is minimized. The approach is reminiscent of online passive-aggressive algorithms [16]. Hence, control laws of higher priority primitives only learn the offset between their desired behavior and the behavior of the lower priority primitives. This structure allows them to override the actions of the primitives of lesser priority and, therefore, add more detailed control in the regions of the state space they are concerned with. The combined control of all primitives is

$$\mathbf{u} = \mathbf{u}_0 + \sum_{n=1}^{N} \Delta \mathbf{u}_n,$$

where $\mathbf{u}_0$ is the null-space control and $\Delta \mathbf{u}_n$ are the offset controls of the N primitives.

Such control laws can be expressed by changing the program in Eq. (5) to

$$\begin{aligned} \min_{\mathbf{u}_i} J &= \left(\mathbf{u}_i - \sum_{j=1}^{i-1} \Delta \mathbf{u}_j - \mathbf{u}_0\right)^{\mathrm{T}} \mathbf{N} \left(\mathbf{u}_i - \sum_{j=1}^{i-1} \Delta \mathbf{u}_j - \mathbf{u}_0\right) \\ s.t.\ \ddot{\mathbf{x}}_i &= \pi_i(\mathbf{x}_i, \dot{\mathbf{x}}_i, z), \end{aligned}$$

where the primitives need to be learned in the increasing order of their priority, the primitive with the lowest priority is learned first, the primitive with the highest priority is learned last. The regression in Eq. (6) changes to

$$\boldsymbol{\theta}_i = \left(\boldsymbol{\Phi}_i^{\mathrm{T}} \hat{\mathbf{W}}_i \boldsymbol{\Phi}_i + \lambda \mathbf{I}\right)^{-1} \boldsymbol{\Phi}_i^{\mathrm{T}} \hat{\mathbf{W}}_i \hat{\mathbf{U}}_i,$$

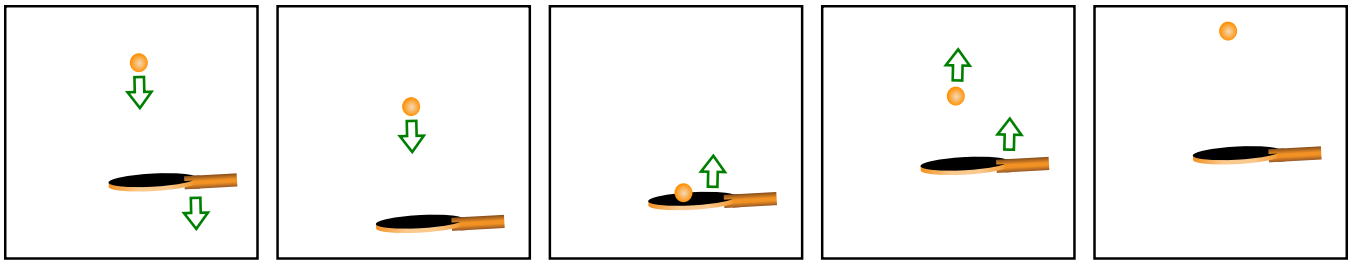
where $\hat{\mathbf{U}}_i$ contains the offset controls $\hat{\mathbf{u}}_{i,t} = \mathbf{u}_{i,t} - \sum_{j=1}^{i-1} \Delta \mathbf{u}_{j,t} - \mathbf{u}_{0,t}$ for all time-steps t, where $\Delta \mathbf{u}_{j,t} = \boldsymbol{\phi}_{i,t}^{\mathrm{T}} \boldsymbol{\theta}_j$. The weighting matrix $\hat{\mathbf{W}}_i$ now has the weights $\hat{w}_t = \exp\left(-\alpha \hat{\mathbf{u}}_{i,t}^{\mathrm{T}} \mathbf{N} \hat{\mathbf{u}}_{i,t}\right)$ on its diagonal and matrix $\hat{\mathbf{U}}_i$ contains offset controls $\hat{\mathbf{u}}_{i,t}$. The kernelized form of the prioritized control law can be obtained analogously. The complete approach is summarized in Algorithm 1.

IV. EVALUATION: BALL-BOUNCING

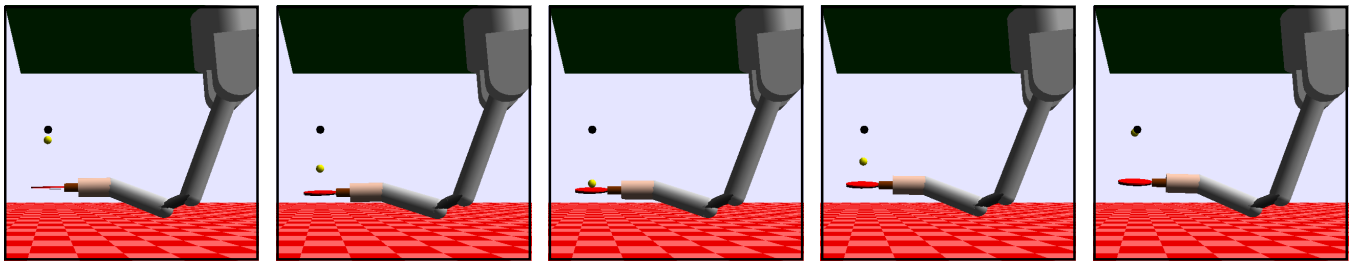
In order to evaluate the proposed prioritized control approach, we chose a ball bouncing task. We describe the task in Section IV-A, explain a possible higher level strategy in Section IV-B, and discuss how the proposed framework can be applied in Section IV-C.

A. Task Description

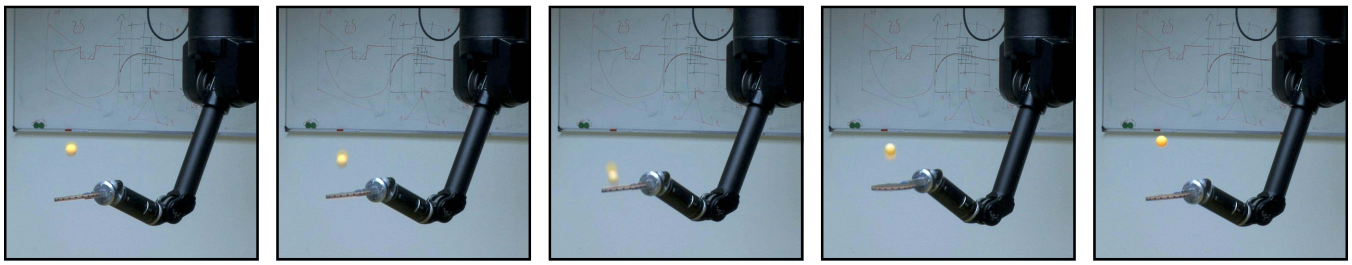
The goal of the task is to bounce a table tennis ball above a racket. The racket is held in the player's hand, or in our case attached to the end-effector of the robot. The ball is supposed to be kept bouncing on the racket. A possible movement is illustrated in Fig. 2.



(a) Exaggerated schematic drawing. The green arrows indicate velocities.



(b) Paddling movement for the simulated robot. The black ball represents the imagined target (see Sect. IV-B)



(c) Paddling movement for the real Barrett WAM.

Fig. 2: This figure illustrates a possible sequence of bouncing the ball on the racket in a schematic drawing, in simulation, and on the real robot.

It is desirable to stabilize the bouncing movement to a strictly vertical bounce, hence, avoiding the need of the player to move a lot in space and, thus, leaving the work space of the robot. The hitting height is a trade-off between having more time until the next hit at the expense of the next hitting position possibly being further away. The task can be sub-dived into three intuitive primitives: hitting the ball upward, moving the racket under the ball before hitting, and changing the orientation of the racket to move the ball to a desired location. A possible strategy is outlined in the next section.

The ball is tracked using a stereo vision setup and its positions and velocities are estimated by a Kalman filter. To initialize the ball-bouncing task, the ball is thrown towards the racket.

B. Bouncing Strategy

The strategy employed to achieve the desired bouncing behavior is based on an imagined target that indicates the desired bouncing height. This target is above the default posture of the racket. The top point of the ball trajectory is supposed to hit this target, and the stable behavior should be a strictly vertical bounce. This behavior can be achieved by defining a hitting plane, i.e., a height at which the ball is always hit (which corresponds to the default posture of the racket). On this hitting plane, the ball is always hit in a manner that the top point of its trajectory corresponds to the height of the target and the next intersection of the ball trajectory with the hitting plane is directly under the target. See Fig. 3 for an illustration.

To achieve this desired ball behavior, the racket is always moved to the intersection point of the ball trajectory and the hitting plane. By choosing the hitting velocity and the orientation of the racket, the velocity and direction of the ball after being hit can be changed. The required hitting velocity and orientation are calculated using a model of the ball and the racket. The ball is modeled as a point mass that moves according to the ballistic flight equations. For the relatively low speeds and small distances air resistance is negligible. The contact with the racket is modeled as a reflection with a restitution factor.

Using this strategy the ball can be brought back to a strictly vertical bouncing behavior with a single hit. However, this method requires the knowledge of the ball position and velocity, as well as a model of the ball behavior. An alternative strategy that stabilizes the behavior in a completely open loop behavior employs a slightly concave paddle shape [17]. A

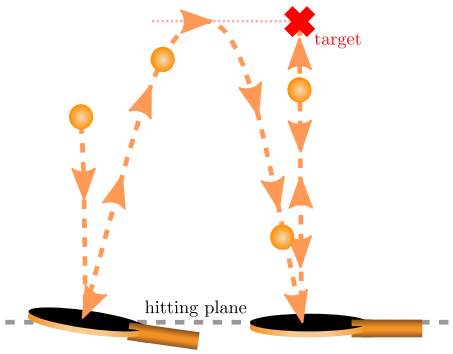


Fig. 3: This figure illustrates the employed strategy for bouncing the ball on the racket. The highest point of the ball trajectory is supposed to coincide with the red target. The racket is always hitting the ball in a fixed height, i.e., the hitting plane. The strategy is to play the ball in a way that the next intersection with the hitting plane is directly below the target and the maximum height of the ball trajectory corresponds to the height of the target. If the bounce works exactly as planned, the ball needs to be hit only once to return to a strictly vertical bouncing behavior.

method similar to the proposed strategy has been employed by[18, 19], and [20] proposed the mirror law for this task. The ball bouncing task has also be employed to study how humans stabilize a rhythmic task [21].

C. Learning Results

As discussed in Section IV-A, the task can be described by three primitives: "move under the ball", "hit the ball", and "change racket orientation". Training data is collected in the relevant state space independently for each primitive. For doing so, the parameters corresponding to the other primitives are kept fixed and variants of the primitive are hence executed from various different starting positions. The primitive "move under the ball" corresponds to movements in the horizontal plane, the primitive "hit the ball" to up and down movements, and the primitive "change racket orientation" only changes the orientation of the end-effector. We collected 30 seconds of training data for each primitive, corresponding to approximately 60 bounces.

Having only three primitives allows it to enumerate all six possible dominance structures, to learn the corresponding prioritized control law, and to evaluate the controller. As intuitive quality measure we counted the number of bounces until the robot missed, either due to imprecise control or due to the ball being outside of the safely reachable work-space.

Table I illustrates the resulting dominance structures. The most relevant primitive is the "hit the ball" primitive, followed by the "move under the ball" primitive. In the table it is clearly visible that inverting the order of two neighboring primitives that are in the preferred dominance order always results in a lower number of hits. Compared to a single model, that was trained using the combined training data of the three primitives, all but two prioritized control laws work significantly better. The ordering may appear slightly

Dominance Structure	Number of Hits	
	in Simulation	on Real Robot
single model	5.70 ± 0.73	1.10 ± 0.99
hit$\succeq$move$\succeq$orient	11.35 ± 2.16	2.30 ± 0.67
hit$\succeq$orient$\succeq$move	10.85 ± 1.46	1.70 ± 0.95
move$\succeq$hit$\succeq$orient	9.05 ± 0.76	1.40 ± 0.70
move$\succeq$orient$\succeq$hit	7.75 ± 1.48	1.40 ± 0.84
orient$\succeq$hit$\succeq$move	5.90 ± 0.85	1.30 ± 0.67
orient$\succeq$move$\succeq$hit	5.35 ± 0.49	1.30 ± 0.48

TABLE I: This table shows the suitability of the possible dominance structures (mean$\pm$std). The "hit the ball" primitive clearly is the dominant one, followed by the "move under the ball" primitive. The prioritized control laws work significantly better than a single model learned using the combined training data of the three primitives. Preliminary results on the real robot confirm this ordering.

counter-intuitive as moving under the ball seems to be the most important primitive in order to keep the ball in the air, allowing for later corrections. However, the robot has a fixed base position and the ball moves quickly out of the safely reachable work-space, resulting in a low number of hits. Additionally, the default position of the racket is almost vertical, hence covering a fairly large area of the horizontal plane resulting in robustness with respect to errors in this primitive.

V. CONCLUSION

In this paper, we have presented a prioritized control learning approach that is based on the superposition of movement primitives. We have introduced a novel framework for learning prioritized control. The controls of the lower priority primitives are fulfilled as long as they lay in the null-space of the higher priority ones and get overridden otherwise. As representation for the primitives, we employ the dynamical systems motor primitives [10, 11], which yield controls in the form of desired accelerations. These primitives are executed separately to collect training data. Local linear models are trained using a weighted regression technique incorporating the various possible dominance structures. In the presented ball bouncing task, the movement is restricted to a space where the controls are approximately linear. Hence, a single linear model per primitive was sufficient. This limitation can be overcome by either considering multiple local linear models [5] or by kernelizing the weighted regression, as described in Sect. III-A and III-B.

The dominance structure of the task was determined by testing all possible structures exhaustively. Intuitively, the lower priority primitives represent a global behavior and the high priority primitives represent specialized corrections, hence overriding the lower priority controls. In most cases, the resulting prioritized control works significantly better than a single layer one that was trained with the combined training data of all primitives. As illustrated by the evaluations, the dominance structure can have a significant influence on the global success of the prioritized control. Enumerating all possible dominance structures is factorial in

the number of primitives and hence unfeasible in practice for more than four primitives. In this case, smarter search strategies are needed.

The success of the different dominance structures not only depends on the task but also on the employed strategy of activating and adapting the different primitives. An interesting area for future research could be to jointly learn the prioritized control and the strategy.

The presented approach has been evaluated both in simulation and on a real Barrett WAM and we have demonstrated that our novel approach can successfully learn a ball-bouncing task.

REFERENCES

[1] J. Peters, M. Mistry, F. Udwadia, J. Nakanishi, and S. Schaal, "A unifying framework for robot control with redundant dofs," *Autonomous Robots*, vol. 24, pp. 1–12, 2008.

[2] F. Chaumette and E. Marchand, "A redundancy-based iterative approach for avoiding joint limits: application to visual servoing," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 5, pp. 719–730, 2001.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[4] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.

[5] J. Peters and S. Schaal, "Learning to control in operational space," *International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, 02 2008.

[6] W. Findeisen, F. N. Bailey, M. Brdeys, K. Malinowski, P. Tatjewski, and J. Wozniak, *Control and coordination in hierarchical systems*, ser. International series on applied systems analysis. Chichester: J. Wiley, 1980.

[7] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *International Journal of Humanoid Robotics*, vol. 2, no. 4, pp. 505–518, 2005.

[8] D. Precup, R. S. Sutton, and S. P. Singh, "Theoretical results on reinforcement learning with temporally abstract options," in *0th European Conference on Machine Learning (ECML)*, 1998, pp. 382–393.

[9] C. Towell, M. Howard, and S. Vijayakumar, "Learning nullspace policies," in *IEEE Int. Conf. Intelligent Robots and Systems*, 2010.

[10] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Washington, DC, May 11-15 2002, pp. 1398–1403.

[11] S. Schaal, P. Mohajerian, and A. J. Ijspeert, "Dynamics systems vs. optimal control - a unifying view," *Progress in Brain Research*, vol. 165, no. 1, pp. 425–445, 2007.

[12] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *Proceedings of the IEEE International Conference on Humanoid Robots (HUMANOIDS)*, 2008, pp. 91–98.

[13] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 853–858.

[14] M. Welling, "The Kalman filter," Lecture Notes, 2010.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer Verlag, 2006.

[16] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *Journal of Machine Learning Research*, vol. 7, pp. 551–585, 2006.

[17] P. Reist and R. D'Andrea, "Bouncing an unconstrained ball in three dimensions with a blind juggling robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 2753–2760.

[18] P. Kulchenko. (2012, Mar.) Robot juggles two ping-pong balls. [Online]. Available: http://notebook.kulchenko.com/juggling/robot-juggles-two-ping-pong-balls

[19] M. Müller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 5113–5120.

[20] M. Buehler, D. E. Koditschek, and P. J. Kindlmann, "Planning and control of robotic juggling and catching tasks," *International Journal of Robotic Research*, vol. 13, no. 2, pp. 101–118, 1994.

[21] S. Schaal, D. Sternad, and C. G. Atkeson, "One-handed juggling: a dynamical approach to a rhythmic movement task," *Journal of Motor Behavior*, vol. 28, no. 2, pp. 165–183, 1996.